Remote Debugging of Embedded System

by

Xiaoxi (Jessie) Zhao

A Dissertation

Submitted in partial fulfillment of the requirements

for Graduate with Honor Distinction in Bachelor of Science Degree in Electrical and Computer Engineering

in the undergraduate college of

The Ohio State University, 2013

Adviser:

Professor Furrukh Khan

Abstract

Microcontrollers (MCUs) conventionally are programmed by connecting the MCU to a debugger then to the computer where a graphical user interface (IDE) is installed. The necessity of wired connection limits the easiness in programming the MCU after it has been installed in a product that may be used at remote or hard to reach locations, like a patient's body or deep inside a volcano. The goal of our research is to address the shortcomings of wired programming and debugging of microcontrollers by designing and implementing a remote debugging framework using WiFi technology so that scientists and engineers can manage, monitor, modify and debug embedded systems (at the register level) from remote locations. In our research, the feasibility of this idea is tested and proven by using the MSP430FR5739 microcontroller (later in this document we will refer to it as FR5739 for short) with CC3000 wireless module and a Graphical User Interface implemented in Windows Presentation Foundation (WPF) using the C# programming language. We were able to communicate register level information between the GUI and FR5739 for debugging purposes which is key to microcontroller debugging. We also developed a method to modify executable memory locations inside the microcontroller to enable programing at the level of machine language.

A prototype of a framework for debugging within Local Area Networks will be proposed and discussed in this thesis. As future work we will extend the framework with web services using Windows Communication Foundation (WCF) to enable remote debugging across cities or continents.

Acknowledgements

Many thanks to my adviser, Furrukh. S. Khan, who has guided me in entering the field of applied software engineering and provided me with valuable knowledge from an experienced programmer's point of view. Thanks to Professor Rajiv Ramanath with whom we have written a joint proposal on Remote Debugging of Embedded Systems, and thanks to Juniper Networks for funding this project. Thanks to the Ohio State University for constantly supporting undergraduate research.

Table of Contents

List of Figures

List of Tables

Part 1. Introduction

Embedded Systems are computers designed for specific control functions within a larger system.[1] There are countless number of distributed embedded systems deployed worldwide by universities, industry and government agencies.

The processing core of embedded systems is either microcontrollers (MCU) or digital signal processors (DSP). The program that determines what the processing core should do, like performing calculations, processing and transferring of data etc, is written typically in assembly or C or a mixture of both, typically utilizing a Graphical User Interface (GUI). The code is later downloaded to the microcontroller after being compiled and debugged via a wired connection to the debugger and to the microcontroller. Even though many of the embedded devices are connected to the Internet, debugging and monitoring of these devices (at the register level) are mostly done locally.

A wired connection has many limitations. For example, if the MCU is embedded inside a patient's body or buried under the ground, you cannot make any changes easily and promptly. The ability to wirelessly debug the microcontroller provides flexibility for researchers and developers to adapt and react to new demands.

This project is a part of a bigger project named Framework for Remote Debugging of Embedded Cloud System jointly proposed by Prof Furrukh Khan, Prof Rajiv Ramnath, Dong Zhang and Xiaoxi Zhao (author of this thesis). In the bigger project we propose to use Juniper Academic Alliance funding and equipment resources to investigate the use of

Juniper technology to address the problem of dynamically adaptive routing in support of application scenarios, and supported by active using network-monitoring tools. Application scenarios, as well as the monitoring tools, being developed at OSU in other research projects (the subject of this thesis) will be made use of in this project.

Part 2. Methodology

2.1 Proposed General Infrastructure for Wireless Debugging

We have selected WiFi[2] as the wireless protocol to use in order to realize wireless debugging. The benefits of selecting WiFi over other wireless protocols is discussed in the discussion section. The general architecture is shown in the following diagram.

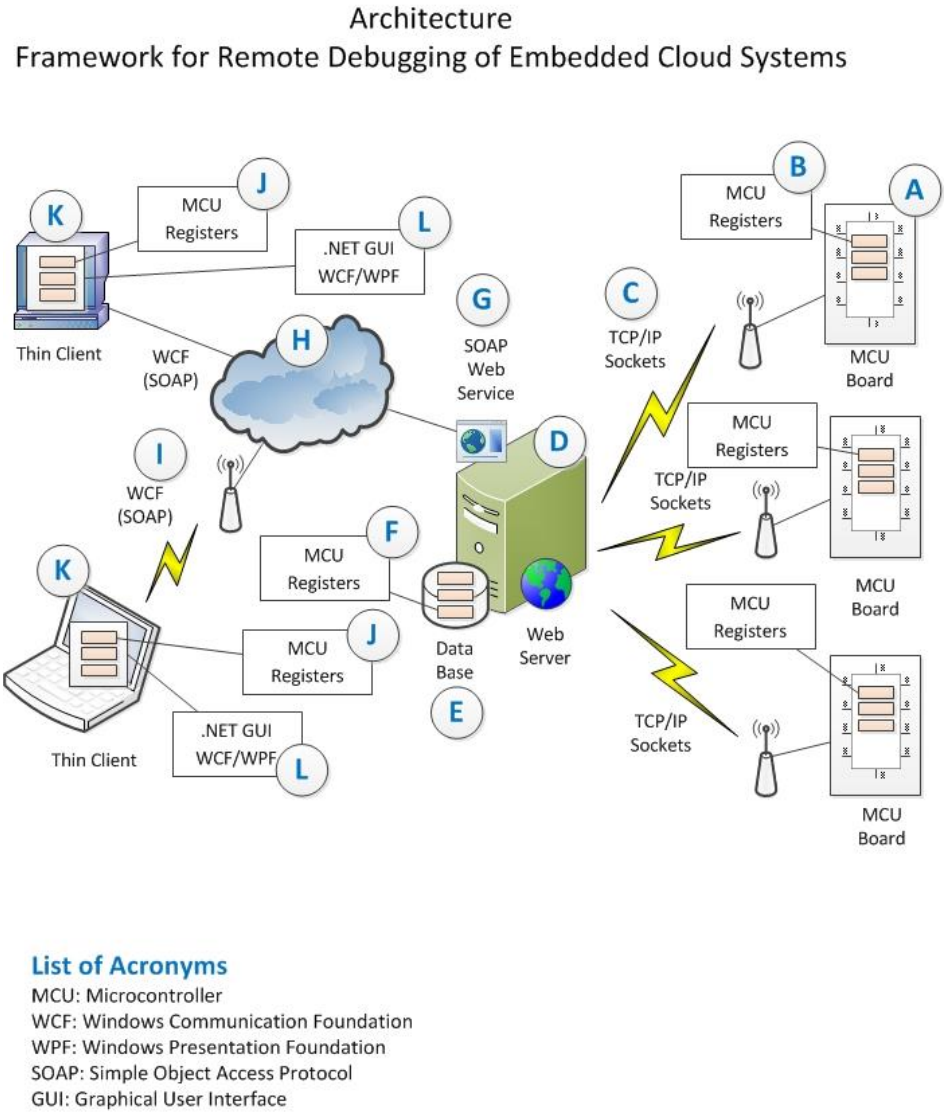


Figure 1. Framework for Remote Debugging of Embedded Cloud Systems Proposed Infrastructure

The aim of the project is to make the registers and memory locations (B) of the distributed microcontrollers (A) accessible (J) to clients (K) connected to the Internet (H). The clients should be able to inspect and change the values of these registers and memory locations by using a GUI (L) which will graphically show the state of the registers (J).

In this infrastructure we propose to accomplish remote debugging by connecting the distributed microcontrollers (A) to a Web Server (D) via. WiFi using TCP/IP sockets programming in C# (.NET 4.5) (C).The microcontrollers (A) will be running compiled embedded C code. We will be using TI's (Texas Instrument's) microcontrollers and for embedded C programming we will use TI's Code Composer Studio. The code on the server (D) and clients (K) will be developed in C# (.NET programming) using Microsoft's Visual Studio IDE (Integrated Development Environment).

The server (D) will store the state of the memories of the microcontrollers in a SQL Server database (E). The data on the database will be made accessible to the cloud through SOAP web services (G) running on the Web Server (D). We will use .NET WCF (Windows Communication Foundation) to build these services and to communicate with them.

The thin client programming on the clients (K) will be done by using C#. WCF (Windows Communication Foundation) will be used to communicate with the web service (G) running on the web server (D). The GUI (L) will be built using WPF (Windows Presentation Foundation). The clients (K) could be hardwired to the cloud or use WiFi (I).

The user will remotely debug the programs running on the microcontrollers (A) by manipulating menus, buttons, text boxes, etc. on the client's GUI (L) by directly interacting with the registers and memory locations (B) of the microcontrollers (A).

2.2 Enable Re-programming using by using Unified FRAM

In order to program a microcontroller, we need to be able to write the executable statements into the memory locations of the MCU without consuming excessive amount of time and power. Typically the main code is located Read Only Memory (ROM). Writing to ROM consumes more time and also requires higher voltage levels, therefore rewriting during execution is made impossible since higher voltages are needed.

In this project we select the Texas Instrument MSP430FR5739 microcontroller which contains 16kB (kilo Bytes) of Ferroelectric Random Access Memory (FRAM). The size of the FRAM memory is large enough to fit most of the code required in our research. More importantly, this FRAM space is unified memory, which means that it can be configured through software as either executable, readable, writable or combinations of these. By locating the main function code (.text region for C compiled code) that will be subject to changes in executable and read/write enabled regions, we are able to change the program code at run time.

Part 3. Results

Currently we have tested debugging within the Local Area Network (parts A, B, C in the framework shown in Figure. 1).

We have accomplished the following:

1. Have the MCU periodically send to the router (working as an Access Point (AP)) data packages containing sensor data and register level information like the Program Counter (PC), Status Register (SR) and the Stack Pointer (SP), which are key registers for debugging a microcontroller using TCP/IP socket communications. The flow chart of TCP/IP communication loop is as shown in Figure. 2. The PC, SR, SP registers are displayed in the WPF GUI for user inspection. Initial version of our GUI is shown in Figure. 3.

   The code on the MCU side is based on the modified sample code from Texas Instruments[2]. This code along with the code for the WPF GUI is attached as appendices A and B.

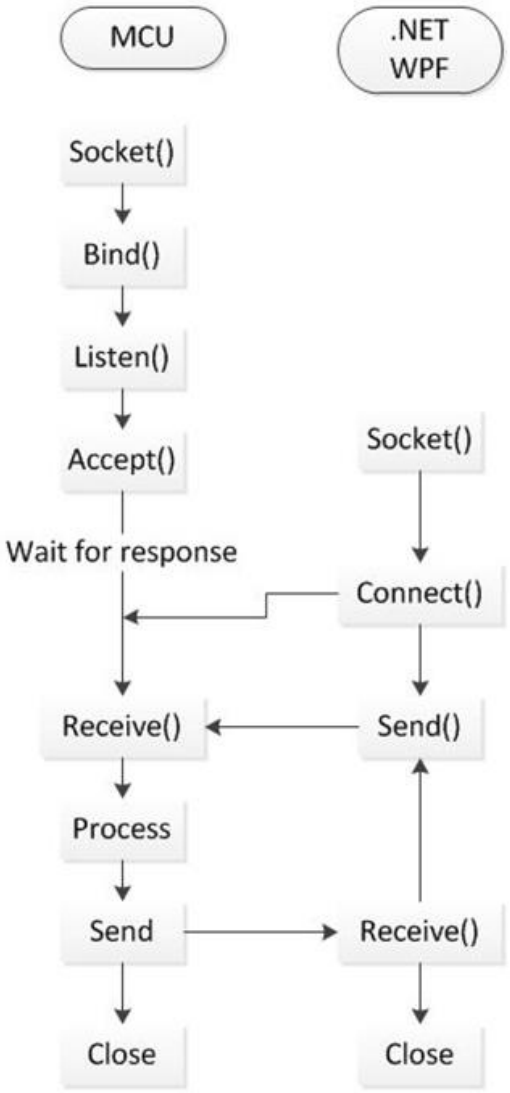


Figure 2. Flow Chart of TCP/IP communication between MCU and computer[3]

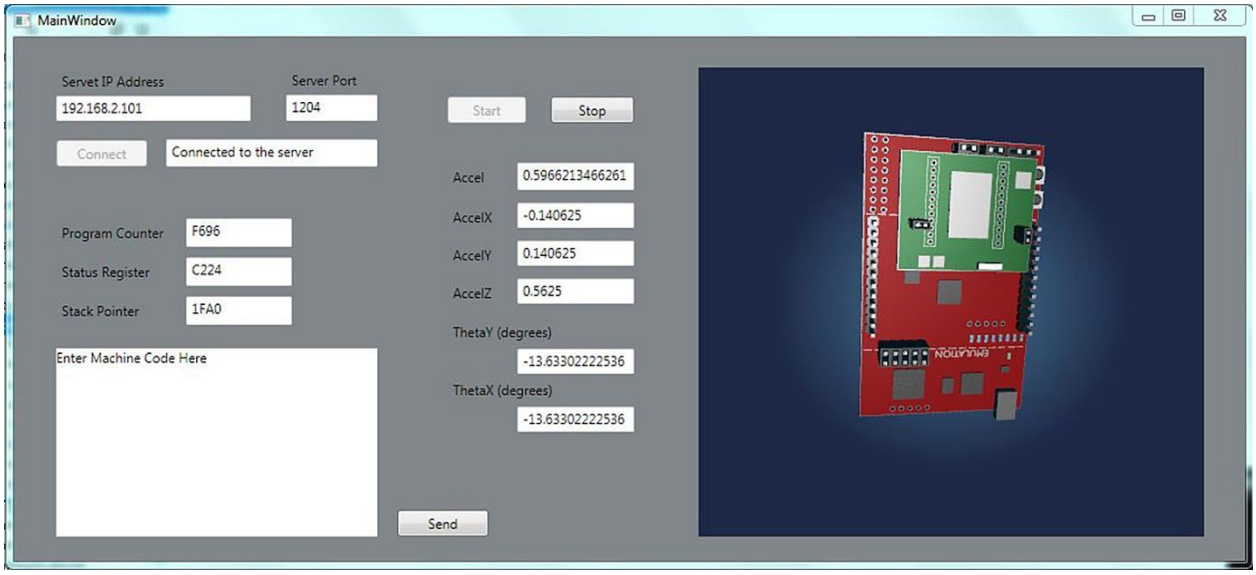
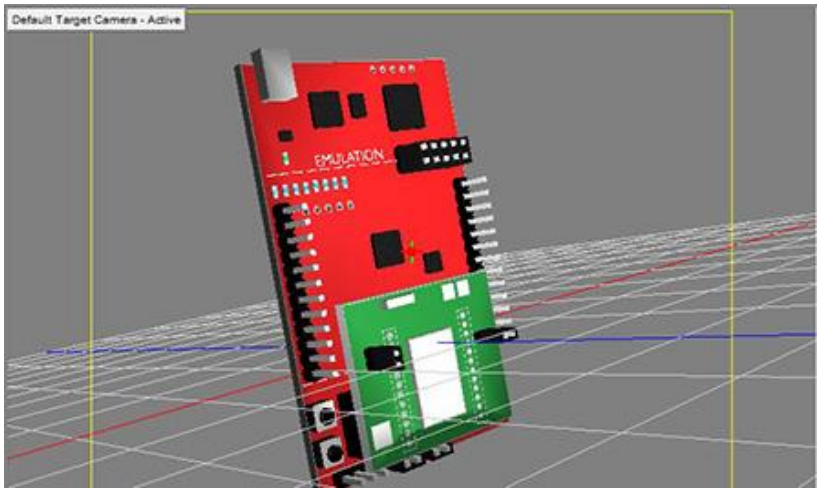


Figure 3. Prototype WPF GUI

2. Have the MCU periodically sending sensor (accelerometer) data to the WPF GUI running on a PC via TCP/IP. A 3D Model of the microcontroller built using ZAM3D is shown in which reflects the actual orientation of the MCU board.



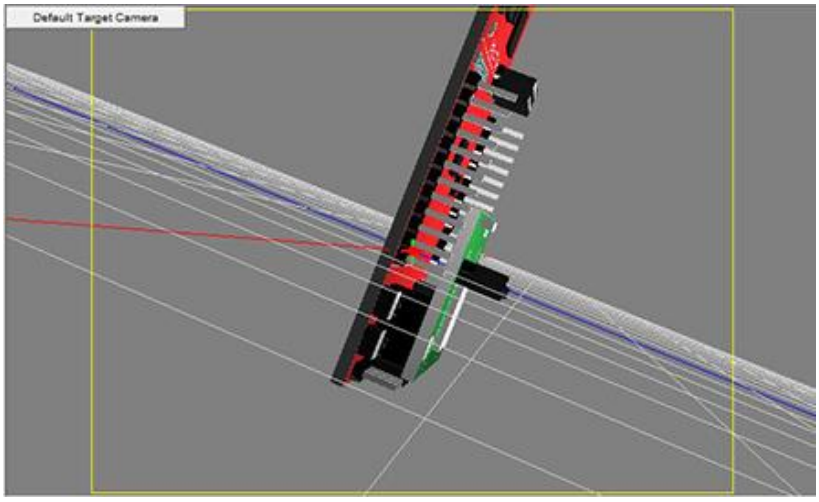


Figure 4. ZAM3D Screenshots of the Microcontroller 3D Model

Demonstrate that code can be modified at run time. This is demonstrated in a sample code in which the program lights up LED8. By changing the actual code in the program region at run time, the code is altered to light up LED7 instead. A reset is generated using the Watch Dog Timer after the code has been altered. The code for this part is attached in appendix C.

3. It was found that with a single Access Point, we were able to perform the above tasks wirelessly to within a radius of about 30m. This value was tested in Caldwell Lab with obstacles between the AP and the MCU. Actual range without obstacles could be longer.

Part 4. Discussion

4.1 Debugging Framework within LAN

4.1.1 Overview of Modules required to support Wireless Debugging

Based on the results we have obtained so far, the preliminary design of our debugging framework within an LAN is shown in Figure. 5.

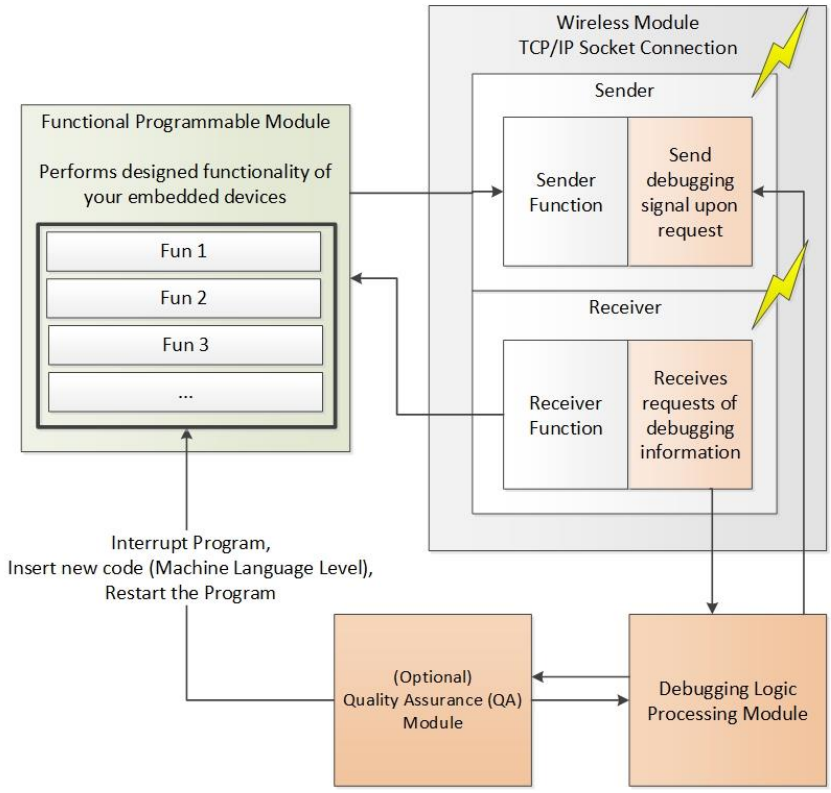


Figure 5. MCU side remote debugging framework

In order to support remote debugging, the MCU side (embedded system) contains three required modules and an optional module. Each module should occupy a continuous block of memory.

1) Functional Programmable Module: Contains functions of the main program to be debugged. The parts of program that are subject to future changes must be contained in RAM, preferably FRAM for low power consumption.
2) Wireless Module: handles sending and receiving of TCP/IP packet to and from the PC. It invokes the debugging module upon reception of debugging requests.
3) Debugging Module: Receives code from wireless module, modifies and inserts the code into the appropriate function location in memory, a restart of the MCU is required after code has been changed.
4) (optional) Quality Assurance Module: Additional RAM testing space. Tests the code in the working environment before the code is put into the Functional Programmable Module by the Debugging Module.

4.2.2 Guidelines for the Functional Programmable Module

There are two working scenarios for re-programmability. First, when the user needs to rewrite the entire functional module. This is needed when the user is still in the design phase. The second scenario is applicable after the design has been accomplished. In that case there is seldom the need to reprogram the entire functional program.

In our framework we think that it is best to structure the program with interfaces that can be called as needed. These interfaces must be located in fixed memory blocks so that changes can be made for specific functions. A list of interfaces and their location and length must be provided as a part of the specifications in order for the user to reach these interfaces wirelessly.

In addition, security checks preventing accidental writing to the wrong interfaces need to be incorporated into the design of the functional module. This part can be added to the quality assurance module.

4.2.3 Guidelines for the Wireless Module

The goal in designing the wireless module is to allow the user to use any wireless module with TCP/IP communication setup as shown on the left hand side of Figure. 2 to support wireless debugging. The wireless module itself should be unaware of what module will handle the data. Currently we suggest using a chain of responsibility pattern with the wireless module.

The data sent from the PC to the receiver needs to be composed of a pre-amble and the actual data. The pre-amble is used to indicate the data packet is for the current MCU functionality or for debugging purposes. After the TCP/IP socket is created and the client's connectivity is confirmed, the wireless module passes the received data to the debugging module. The debugging module checks the pre-amble and if the data pre-amble matches, the module will go ahead and process the data. Otherwise the data is passed on to the functional module.

The advantage of this is to keep the wireless module as universal and simple as possible. The type of pre-amble can be chosen and changed by the user without modifying the wireless module.

4.2.4 Guidelines for Debugging Logic

In the debugging logic part we need a mix of C and assembly code to enable lower level control of the MCU. A memory map of the functional module is required to perform the corresponding debugging. Data sent as debugging code needs to contain information about where the code should be located, length of memory space needed to be changed and the corresponding machine code instruction. With the help of assembly code we can put the machine code in the specified location. A reset is needed after the code has been in place to start the new function written. (The feasibility has been proven possible in our project)

Certain data package format/protocol need to be further evaluated for maximum reliability.

4.2 WiFi vs ZigBee and Bluetooth in Remote Debugging Design

A table comparing the key characteristics of ZigBee, Wi-Fi and Bluetooth[3] is summarized below

Table 2. The key characteristics of Zigbee, Wi-Fi and Bluetooth[4]

	ZigBee	Wi-Fi	Bluetooth
Range	10-100 meters	50-100 meters	10 – 100 meters
Networking Topology	Ad-hoc, peer to peer, star, or mesh	Point to hub	Ad-hoc, very small networks
Operating Frequency	868 MHz (Europe) 900-928 MHz (NA), 2.4 GHz (worldwide)	2.4 and 5 GHz	2.4 GHz

Complexity (Device and application impact)	Low	High	High
Power Consumption (Battery option and life)	Very low (low power is a design goal)	High	Medium
Security	128 AES plus application layer security		64 and 128 bit encryption
Typical Applications	Industrial control and monitoring, sensor networks, building automation, home control and automation, toys, games	Wireless LAN connectivity, broadband Internet access	Wireless connectivity between devices such as phones, PDA, laptops, headsets

Many embedded systems use ZigBee or Bluetooth as their remote access protocol because ZigBee is simpler and consumes less power compared with WiFi. However in order to have secure and full access to the powerful and flexible TCP/IP stack (802.11) WiFi is the only solution.

4.3 Future Developments

Future development of this project consists of four parts.

1) Test and improve the framework described in the discussion section. The issue of reusability needs to be addressed.

2) Build web services on top of the LAN part to enable remote access across cities, countries etc. We propose to use Windows Communication Foundation for this part. A framework for web services side needs to be added to the LAN framework.
3) Integrate the web services with the LAN part and test the entire remote debugging framework.
4) Improve the graphical user interface

Part 5. Conclusions

To conclude, we find that remote debugging of embedded system is possible with unified memory FRAM microcontrollers, Windows Presentation Foundation (WPF) and TCP/IP protocol. The most critical portion of the project, which is the debugging loop, has been tested for feasibility. We have come up with a preliminary framework that sets a guidelines for a fully functional remote debugging process which also addresses reusability. Future work needs to be done including improvement of the preliminary framework and integration of the LAN part with web services for true remote access.

Reference

[1] Michael Barr. "Embedded Systems Glossary". *Neutrino Technical Library*. Retrieved 2007-04-21.

[2] "CC3000 Wi-Fi MSP430 FRAM Sensor Application Release Notes." - *Texas Instruments Embedded Processors Wiki*. Texas Instrument, 17 Feb. 2013. Web. 24 Apr. 2013

[3] Velte, Toby J., and Anthony T. Velte. "*Cisco 802.11 Wireless Networking Quick Reference*". Indianapolis, IN: Cisco, 2006. Print.

[4] "SENA BLOG." *SENA BLOG*. N.p., 25 Feb. 2010. Web. 24 Apr. 2013.

Appendices

Appendix A – MCU side code

The code is modified based upon Texas Instrument sensor application sample code (reference 2). The demo.c part and server.c part has been modified as following.

demo.c

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

using System.Threading;
using System.Net;
using System.Net.Sockets;

using System.ComponentModel;
namespace WpfApplication2
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {

        private Socket sock = null;
        private Boolean runFlag = false;
        private delegate void NoArgDelegate();
        private byte[] sensorBuffer = { 128, 0xBE, 128, 128, 128, 70, 36, 0x11, 0x22,
0x33, 0x44, 0x55, 0x66, 0xEF };
        private double accelX, accelY, accelZ;
        private string PC;
        private string SR;
        private string SP;
        private double angleX, angleY;
        private double[] accelXhist = new double[] { 0, 0, 0, 0, 0 };
        private double[] accelYhist = new double[] { 0, 0, 0, 0, 0 };
        private double[] accelZhist = new double[] { 0.6, 0.6, 0.6, 0.6, 0.6 };
        private int ACCEL_BUFFER_SIZE = 5;
        private System.Windows.Media.Media3D.Vector3D dir1 = new
System.Windows.Media.Media3D.Vector3D(0, 0, -1);
```

```
        private System.Windows.Media.Media3D.Vector3D dir2 = new
System.Windows.Media.Media3D.Vector3D(0, 0, 1);
        private String server = "192.168.2.100";
        private int servPort = 1204;
        private byte[] byteBuffer = Encoding.ASCII.GetBytes("DATA");

    //    private byte[] byteSend = Encoding.ASCII.GetBytes("ABCD");
        private byte[] byteSend = Encoding.ASCII.GetBytes("TEST");
 //       private int[] testPC = new int[4]{0xA, 0xB, 0xC, 0xD};
 //      private byte[] testByte;

       // private byte[] byteSend;
       // private int PCInstructTest = 0xABCD;

        private int i;



        public MainWindow()
        {
            InitializeComponent();

            //sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
            //Msg.Text = "Connecting to the server";
            //sock.Connect(server, servPort);
            //sock.Send(byteBuffer, 0, byteBuffer.Length, SocketFlags.None);
            IP_TextBox.Text = server;
            Port.Text = servPort.ToString();
            Stop.IsEnabled = false;
        }


        private void Connect_Click(object sender, RoutedEventArgs e)
        {
            Msg.Text = "Attempting to conenct to the server ...";
            String server = this.IP_TextBox.Text;
            sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);

            sock.Connect(server, servPort);
            sock.Send(byteBuffer, 0, byteBuffer.Length, SocketFlags.None);
            Msg.Text = "Connected to the server";
            Connect.IsEnabled = false;
        }

        private void Start_Click_1(object sender, RoutedEventArgs e)
        {
            Start.IsEnabled = false;
            Stop.IsEnabled = true;
            Camera.LookDirection = new System.Windows.Media.Media3D.Vector3D(0, 0, -1);

            (new Action(this.DoStuff)).BeginInvoke(null, null);

        }

        public void DoStuff()
        {
```

```
            runFlag = true;
            i = 0;
            while (runFlag)
            {
                ++i;

                int bytesRcvd = sock.Receive(sensorBuffer, 0, sensorBuffer.Length,
SocketFlags.None);
                this.Dispatcher.Invoke(System.Windows.Threading.DispatcherPriority.Normal,
                    new NoArgDelegate(doStuff));

            }
            sock.Close();
        }

        private void doStuff()
        {
            accelX = sensorBuffer[2];
            accelY = sensorBuffer[3];
            accelZ = sensorBuffer[4];
            PC = sensorBuffer[7].ToString("X") + sensorBuffer[8].ToString("X");
            SR = sensorBuffer[9].ToString("X") + sensorBuffer[10].ToString("X");
            SP = sensorBuffer[11].ToString("X") + sensorBuffer[12].ToString("X");
            accelX = accelX - 64;
            accelY = accelY - 64;
            accelZ = accelZ - 64;
            accelX = accelX * 3.0 / 64.0;
            accelY = accelY * 3.0 / 64.0;
            accelZ = accelZ * 3.0 / 64.0;


            //averaging filter
            for (int ind=0; ind< (ACCEL_BUFFER_SIZE-1); ind++) {
                accelXhist[ind] = accelXhist[ind+1];
                accelYhist[ind] = accelYhist[ind+1];
                accelZhist[ind] = accelZhist[ind+1];
            }
            accelXhist[ACCEL_BUFFER_SIZE-1] = accelX;
            accelYhist[ACCEL_BUFFER_SIZE-1] = accelY;
            accelZhist[ACCEL_BUFFER_SIZE-1] = accelZ;
            accelX = 0;
            accelY = 0;
            accelZ = 0;
            for (int ind=0; ind<ACCEL_BUFFER_SIZE; ind++) {
                accelX += accelXhist[ind];
                accelY += accelYhist[ind];
                accelZ += accelZhist[ind];
            }
            accelX = -accelX / ACCEL_BUFFER_SIZE;
            accelY = -accelY / ACCEL_BUFFER_SIZE;
            accelZ = accelZ / ACCEL_BUFFER_SIZE;

            double mag = Math.Sqrt(accelX * accelX + accelY * accelY + accelZ * accelZ);

            angleX = -1.0*Math.Asin(accelY / mag) * 180.0 / Math.PI;
         /*  if (accelZ < 0.0)
            {
               if(accelY > 0.0)
```

```
            {
             angleX = -angleX - 180.0;
            }
            if (accelY < 0.0)
            {
                angleX = -angleX + 180.0;
            }

            if (accelX > 0.0)
            {
                angleY = -angleY + 180.0;
            }
            if (accelX < 0.0)
            {
                angleY = -angleY - 180.0;
            }

        } */

        if (accelZ < 0.0)
        {
            light.Direction = dir2;
        }
        else
        {
            light.Direction = dir1;
        }
        angleY = Math.Asin(accelX / mag) * 180.0 / Math.PI;

        AccX.Text = accelX.ToString();
        AccY.Text = accelY.ToString();
        AccZ.Text = accelZ.ToString();
        Acc.Text = mag.ToString();
        ThetaX.Text = angleX.ToString();
        ThetaY.Text = angleY.ToString();
        PC_Textbox.Text = PC;
        SR_Textbox.Text = SR;
        SP_Textbox.Text = SP;
    }


    private void Button_Click_2(object sender, RoutedEventArgs e)
    {
        Stop.IsEnabled = false;
        runFlag = false;

        //sock.Close();
    }

    private void ScrollBar_ValueChanged_1(object sender,
RoutedPropertyChangedEventArgs<double> e)
    {

    }
```

```
}

public class Board : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected void Notify(string propName)
    {
        if (this.PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propName));
        }
    }

    private double _AccX;
    public double AccX
    {
        get { return _AccX; }
        set
        {
            _AccX = value;
            Notify("AccX");
        }

    }

    private double _AccY;
    public double AccY
    {
        get { return _AccY; }
        set
        {
            _AccY = value;
            Notify("AccY");
        }

    }

    private double _AccZ;
    public double AccZ
    {
        get { return _AccZ; }
        set
        {
            _AccZ = value;
            Notify("AccZ");
        }

    }

}

public class NumConversion
{
    public static byte[] StringToByteArray(string hex)
    {
        return Enumerable.Range(0, hex.Length)
                         .Where(x => x % 2 == 0)
```

```
                            .Select(x => Convert.ToByte(hex.Substring(x, 2), 16))
                            .ToArray();
        }
    }

}
```

Server.c

```
/*****************************************************************************
*
*  server.c - CC3000 Sensor Application Server functionality implementation
*  Copyright (C) 2011 Texas Instruments Incorporated - http://www.ti.com/
*
*  Modified by: Xiaoxi Zhao, The Ohio State University
*
*  Redistribution and use in source and binary forms, with or without
*  modification, are permitted provided that the following conditions
*  are met:
*
*    Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
*    Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the
*    distribution.
*
*    Neither the name of Texas Instruments Incorporated nor the names of
*    its contributors may be used to endorse or promote products derived
*    from this software without specific prior written permission.
*
*  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
*  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
*  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
*  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
*  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
*  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
*  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
*  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
*  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
*  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
*  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*****************************************************************************/

#include "cc3000.h"
#include "msp430fr5739.h"
#include "wlan.h"
#include "evnt_handler.h"    // callback function declaration
#include "nvmem.h"
#include "socket.h"
#include "common.h"
#include "netapp.h"
```

```
#include "board.h"
#include "common.h"
#include "string.h"
#include "demo_config.h"
#include "uart.h"
#include "sensors.h"
#include "board.h"
#include "terminal.h"
#include "strlib.h"
#include "server.h"

long serverSocket;
sockaddr serverSocketAddr;

/** \brief Definition of data packet to be sent by server */
//unsigned char sensorDataPack[] = { '\r', 0xBE, 128, 128, 128, 70, 36, 0xEF };
//unsigned char debugInfoPackp[] = { '\r', 0xBE, 0xFF, 0xFF, 0xEF };
unsigned char receiveBuffer[SERVER_RECV_BUF_SIZE];

char serverErrorCode = 0;

//*****************************************************************************
//
//! Initialize Connection Server
//!
//! \param  none
//!
//! \return none
//!
//! \brief  Waits for a connection where we will
//!
//*****************************************************************************
void initServer(void)
{
    char portStr[12];
    memset(portStr,0,sizeof(portStr));

    // Open Server Socket
    serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    if (serverSocket == -1)
    {
        serverError(SERV_ERR_SOCKET);
        wlan_stop();
        while(1)
        {
            __no_operation();
        }
    }

    // Open port
    int port = SERVER_PORT;
    serverSocketAddr.sa_family = AF_INET;

    // Set the Port Number
    serverSocketAddr.sa_data[0] = (port & 0xFF00)>> 8;
    serverSocketAddr.sa_data[1] = (port & 0x00FF);
```

```
    memset (&serverSocketAddr.sa_data[2], 0, 4);

    if (bind(serverSocket, &serverSocketAddr, sizeof(sockaddr)) != 0)
    {
        serverError(SERV_ERR_BIND);
        return;
    }


    // Start Listening
    if (listen (serverSocket, 1) != 0)
    {
        serverError(SERV_ERR_LISTEN);
        return;
    }

    setCC3000MachineState(CC3000_SERVER_INIT);
    terminalPrint("Server Initialized on port ");
    itoa(port, portStr, DECIMAL_BASE);
    terminalPrint(portStr);
    terminalPrint("\r\n");

}

//*****************************************************************************
//
//! \brief  Waits and handle a client connection
//!
//! \param  none
//!
//! \return none
//!
//
//*****************************************************************************
void waitForConnection(unsigned char* dataPackPointer, int dataPackSize)
{
    sockaddr clientaddr;
    socklen_t addrlen;
    int clientDescriptor = -1;
    volatile int bytesRecvd = 0;
    volatile int RecvTest = 0;
    int curSocket = 0;
    int bytesSent = 0;
    int optval, optlen;
    char clientIP[4];

    // Check whether the server functionality is successfully initialized
    if(currentCC3000State() & CC3000_SERVER_INIT)
    {
        // Begin waiting for client and handle the connection
        while(1)
        {
            //
            // Handle any un-solicited event if required - the function shall be
triggered
            // few times in a second
            //
            hci_unsolicited_event_handler();
```

```
            addrlen = sizeof(clientaddr);

            terminalPrint("Waiting for Clients\r\n");
            // accept blocks until we receive a connection
                                        while ( (clientDescriptor == -1) ||
(clientDescriptor == -2) )
                                        {
                clientDescriptor = accept(serverSocket, (sockaddr *) &clientaddr,
&addrlen);
                                        }

            //
            // Handle any un-solicited event if required - the function shall be
triggered
            // few times in a second
            //
            hci_unsolicited_event_handler();

            // Call user specified Clietn Accepted Event Handler

            if(clientDescriptor >= 0)
            {
                setCC3000MachineState(CC3000_CLIENT_CONNECTED);
                terminalPrint("Client ");
                // Read IP in reverse due to endianess and print
                clientIP[0] = clientaddr.sa_data[5];
                clientIP[1] = clientaddr.sa_data[4];
                clientIP[2] = clientaddr.sa_data[3];
                clientIP[3] = clientaddr.sa_data[2];

                printIpAddr(clientIP);
                terminalPrint(" Connected\r\n");
                // Connection Accepted, Wait for data exchange

                char requestBuffer[SERVER_RECV_BUF_SIZE];

                bytesRecvd = recv(clientDescriptor, requestBuffer, sizeof(requestBuffer),
0);

                // Check whether we received the DATA command from the client
                // telling us to begin sending data to it.
                if(strncmp(requestBuffer,"DATA",strlen("DATA")) == 0)
                {
                    while(currentCC3000State() & CC3000_CLIENT_CONNECTED)
                    {
                        // Start Sending Data to server
                        // Send data to CC3000
                        hci_unsolicited_event_handler();
                        unsolicicted_events_timer_disable();
                        toggleLed(CC3000_SENDING_DATA_IND);

                        //RecvTest = recv(clientDescriptor, receiveBuffer,
sizeof(receiveBuffer), 0);

                        //asm("PCTEST: nop");
                        //asm("LABEL:    mov.w  &receiveBuffer, R15");
```

```
/*                        SetupAccel();
                        SetupThermistor();

                        SetupAccelXXX();
                        TakeADCMeas(X_AXIS_MEAS);      // Take 1 ADC Sample for X-AXIS


                        SetupAccelYYY();
                        TakeADCMeas(Y_AXIS_MEAS);      // Take 1 ADC Sample for Y-AXIS


                        SetupAccelZZZ();
                        TakeADCMeas(Z_AXIS_MEAS);      // Take 1 ADC Sample for Y-AXIS


                        SetupThermistorADC();  // One time setup and calibration
                        TakeADCMeas(TEMP_MEAS);         // Take 1 ADC Sample FOR TEMP

                        SetupVcc();
                        TakeADCMeas(VCC_MEAS);

*/
                        __no_operation();

                        bytesSent = send(clientDescriptor, dataPackPointer, dataPackSize,
0);



                        // RecvTest = recv(clientDescriptor, receiveBuffer,
sizeof(receiveBuffer), 0);

                        if (bytesSent != dataPackSize)
                        {
                            // Check if socket is still available
                            curSocket =  getsockopt(clientDescriptor, SOL_SOCKET,
SOCK_DGRAM , &optval, (socklen_t*)&optlen);
                            if (curSocket != 0)
                            {
                                closesocket(clientDescriptor);
                                terminalPrint("Client Disconnected\r\n");

                    clientDescriptor = -1;
                                unsetCC3000MachineState(CC3000_CLIENT_CONNECTED);
                            }
                        }
                        unsolicicted_events_timer_init();
                        __delay_cycles(24000);                //this should wait a second
                    }
                }
                __delay_cycles(1000);
            }
            else if(clientDescriptor == SOCKET_INACTIVE_ERR)
            {
                terminalPrint("Socket Server Timeout. Restarting Server\r\n");
                                                    clientDescriptor = -1;
                // Reinitialize the server
                shutdownServer();
```

```
                initServer();
            }
            hci_unsolicited_event_handler();
        }
    }
}

//*****************************************************************************
//
//! Shut down server sockets
//!
//! \param  none
//!
//! \return none
//!
//! \brief  Waits for a connection where we will
//
//*****************************************************************************
void shutdownServer()
{
     // Close the Server's Socket
    closesocket(serverSocket);
    serverSocket = 0xFFFFFFFF;
}

//*****************************************************************************
//
//! \brief  Waits for a connection where we will
//!
//! \param  none
//!
//! \return none
//!
//
//*****************************************************************************
void serverError(char err)
{
     switch(err)
     {
        case SERV_ERR_SOCKET:
            serverErrorCode = SERV_ERR_SOCKET;
         break;
        case SERV_ERR_BIND:
            serverErrorCode = SERV_ERR_BIND;
         break;
        case SERV_ERR_LISTEN:
            serverErrorCode = SERV_ERR_LISTEN;
            break;
     }
     while(1)
     {
         __no_operation();
     }
}
```

Appendix B – WPF GUI code

XAML part is too large to be included in the thesis. Please contact zhao.304@osu.edu for detailed code.

MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

using System.Threading;
using System.Net;
using System.Net.Sockets;

using System.ComponentModel;
namespace WpfApplication2
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {

        private Socket sock = null;
        private Boolean runFlag = false;
        private delegate void NoArgDelegate();
        private byte[] sensorBuffer = { 128, 0xBE, 128, 128, 128, 70, 36, 0x11, 0x22,
0x33, 0x44, 0x55, 0x66, 0xEF };
        private double accelX, accelY, accelZ;
        private string PC;
        private string SR;
        private string SP;
        private double angleX, angleY;
        private double[] accelXhist = new double[] { 0, 0, 0, 0, 0 };
        private double[] accelYhist = new double[] { 0, 0, 0, 0, 0 };
        private double[] accelZhist = new double[] { 0.6, 0.6, 0.6, 0.6, 0.6 };
        private int ACCEL_BUFFER_SIZE = 5;
        private System.Windows.Media.Media3D.Vector3D dir1 = new
System.Windows.Media.Media3D.Vector3D(0, 0, -1);
        private System.Windows.Media.Media3D.Vector3D dir2 = new
System.Windows.Media.Media3D.Vector3D(0, 0, 1);
        private String server = "192.168.2.100";
```

```
        private int servPort = 1204;
        private byte[] byteBuffer = Encoding.ASCII.GetBytes("DATA");

    //    private byte[] byteSend = Encoding.ASCII.GetBytes("ABCD");
        private byte[] byteSend = Encoding.ASCII.GetBytes("TEST");
 //       private int[] testPC = new int[4]{0xA, 0xB, 0xC, 0xD};
 //      private byte[] testByte;

       // private byte[] byteSend;
       // private int PCInstructTest = 0xABCD;

        private int i;



        public MainWindow()
        {
            InitializeComponent();

            //sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
            //Msg.Text = "Connecting to the server";
            //sock.Connect(server, servPort);
            //sock.Send(byteBuffer, 0, byteBuffer.Length, SocketFlags.None);
            IP_TextBox.Text = server;
            Port.Text = servPort.ToString();
            Stop.IsEnabled = false;
        }


        private void Connect_Click(object sender, RoutedEventArgs e)
        {
            Msg.Text = "Attempting to conenct to the server ...";
            String server = this.IP_TextBox.Text;
            sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);

            sock.Connect(server, servPort);
            sock.Send(byteBuffer, 0, byteBuffer.Length, SocketFlags.None);
            Msg.Text = "Connected to the server";
            Connect.IsEnabled = false;
        }

        private void Start_Click_1(object sender, RoutedEventArgs e)
        {
            Start.IsEnabled = false;
            Stop.IsEnabled = true;
            Camera.LookDirection = new System.Windows.Media.Media3D.Vector3D(0, 0, -1);

            (new Action(this.DoStuff)).BeginInvoke(null, null);

        }

        public void DoStuff()
        {
            runFlag = true;
            i = 0;
            while (runFlag)
```

```
            {
                ++i;

                int bytesRcvd = sock.Receive(sensorBuffer, 0, sensorBuffer.Length,
SocketFlags.None);
                this.Dispatcher.Invoke(System.Windows.Threading.DispatcherPriority.Normal,
                    new NoArgDelegate(doStuff));

            }
            sock.Close();
        }

        private void doStuff()
        {
            accelX = sensorBuffer[2];
            accelY = sensorBuffer[3];
            accelZ = sensorBuffer[4];
            PC = sensorBuffer[7].ToString("X") + sensorBuffer[8].ToString("X");
            SR = sensorBuffer[9].ToString("X") + sensorBuffer[10].ToString("X");
            SP = sensorBuffer[11].ToString("X") + sensorBuffer[12].ToString("X");
            accelX = accelX - 64;
            accelY = accelY - 64;
            accelZ = accelZ - 64;
            accelX = accelX * 3.0 / 64.0;
            accelY = accelY * 3.0 / 64.0;
            accelZ = accelZ * 3.0 / 64.0;


            //averaging filter
            for (int ind=0; ind< (ACCEL_BUFFER_SIZE-1); ind++) {
                accelXhist[ind] = accelXhist[ind+1];
                accelYhist[ind] = accelYhist[ind+1];
                accelZhist[ind] = accelZhist[ind+1];
            }
            accelXhist[ACCEL_BUFFER_SIZE-1] = accelX;
            accelYhist[ACCEL_BUFFER_SIZE-1] = accelY;
            accelZhist[ACCEL_BUFFER_SIZE-1] = accelZ;
            accelX = 0;
            accelY = 0;
            accelZ = 0;
            for (int ind=0; ind<ACCEL_BUFFER_SIZE; ind++) {
                accelX += accelXhist[ind];
                accelY += accelYhist[ind];
                accelZ += accelZhist[ind];
            }
            accelX = -accelX / ACCEL_BUFFER_SIZE;
            accelY = -accelY / ACCEL_BUFFER_SIZE;
            accelZ = accelZ / ACCEL_BUFFER_SIZE;

            double mag = Math.Sqrt(accelX * accelX + accelY * accelY + accelZ * accelZ);

            angleX = -1.0*Math.Asin(accelY / mag) * 180.0 / Math.PI;
         /*  if (accelZ < 0.0)
            {
               if(accelY > 0.0)
               {
                angleX = -angleX - 180.0;
               }
```

```
                if (accelY < 0.0)
                {
                    angleX = -angleX + 180.0;
                }

                if (accelX > 0.0)
                {
                    angleY = -angleY + 180.0;
                }
                if (accelX < 0.0)
                {
                    angleY = -angleY - 180.0;
                }

            } */

            if (accelZ < 0.0)
            {
                light.Direction = dir2;
            }
            else
            {
                light.Direction = dir1;
            }
            angleY = Math.Asin(accelX / mag) * 180.0 / Math.PI;

            AccX.Text = accelX.ToString();
            AccY.Text = accelY.ToString();
            AccZ.Text = accelZ.ToString();
            Acc.Text = mag.ToString();
            ThetaX.Text = angleX.ToString();
            ThetaY.Text = angleY.ToString();
            PC_Textbox.Text = PC;
            SR_Textbox.Text = SR;
            SP_Textbox.Text = SP;
        }


        private void Button_Click_2(object sender, RoutedEventArgs e)
        {
            Stop.IsEnabled = false;
            runFlag = false;

            //sock.Close();
        }

        private void ScrollBar_ValueChanged_1(object sender,
RoutedPropertyChangedEventArgs<double> e)
        {

        }




    }

    public class Board : INotifyPropertyChanged
```

```
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected void Notify(string propName)
    {
        if (this.PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propName));
        }
    }

    private double _AccX;
    public double AccX
    {
        get { return _AccX; }
        set
        {
            _AccX = value;
            Notify("AccX");
        }

    }

    private double _AccY;
    public double AccY
    {
        get { return _AccY; }
        set
        {
            _AccY = value;
            Notify("AccY");
        }

    }

    private double _AccZ;
    public double AccZ
    {
        get { return _AccZ; }
        set
        {
            _AccZ = value;
            Notify("AccZ");
        }

    }

}

public class NumConversion
{
    public static byte[] StringToByteArray(string hex)
    {
        return Enumerable.Range(0, hex.Length)
                         .Where(x => x % 2 == 0)
                         .Select(x => Convert.ToByte(hex.Substring(x, 2), 16))
                         .ToArray();
    }
```

```
    }

}
```

Appendix C – Test Code for Rewriting Executable Code at Run Time

Main.c

```
/*
 * main.c
 */
#include <msp430fr5739.h>
extern void testCodeUpdate(char *debugDataPack, int Val);
extern void AsmPart(void);
char debugDataPack[] = {0x11, 0x22, 0x33, 0x44, 0x55, 0x66};
int valtest;
int valtest2;

void main(void)
{
       WDTCTL = WDTPW + WDTHOLD;

    AsmPart();

//     testCodeUpdate(debugDataPack,sizeof(debugDataPack));

//     valtest = sizeof(debugDataPack);

       AsmPart();

       while(1){}
}

void testCall(void)
{
//     asm(" mov.w  #0x0040, &DebugDataRecv");

       P3OUT ^= (BIT7);
}
```

Test.asm

```
                    .cdecls C,LIST, "msp430fr5739.h"

                    .ref   testCall
                    .def   AsmPart
                    .def   testCodeUpdate

                    .data
DebugDataRecv:      .space 2
DebugInstruct:  .space  6
test:               .word  2
```

```
AsmPart:
                    bis.b  #BIT7, &P3DIR
                    bis.b  #BIT7, &P3OUT



                    mov.w         #0x0040, &0xC20C

                    mov.w  #0, &WDTCTL          ; Software reset

                    ret

testCodeUpdate:

                    mov.w  0(R12), &DebugInstruct
                    mov.w  2(R12), &DebugInstruct+2
                    mov.w  4(R12), &DebugInstruct+4

;                   mov.w  R14, R13


                    ret
                    .end
```